REMARKS

Claims 13-37 are pending in the application at the time of the Office Action. Claims 13-37 are rejected under 35 U.S.C. 103. By this response, Applicant has amended claims 13, 20, 31 and 34. Applicant respectfully submits that the amendment to the claims are based in the specification as originally filed and that no new matter has been added. Entry of the claim amendments is respectfully requested. As such, claims 13-37 are presented for the Examiner's consideration in light of the following remarks.

Reconsideration and allowance of the application is respectfully requested in view of the above amendments to the claims and the following remarks. Applicant requests that the Examiner carefully review any references discussed below to ensure that Applicant's understanding and discussion of the references, if any, is consistent with the Examiner's understanding. For the Examiner's convenience and reference, Applicant's remarks are presented in the order in which the corresponding issues were raised in the Office Action.

A. Examiner Telephone Interview

Applicant(s) and applicant's attorney express appreciation to the Examiner for the courtesies extended during the recent interview held on December 19, 2007. In the interview, Applicant agreed to make clarification amendments to the claims, which Examiner believed would further allowance of the claims. This response includes the substance of the Interview.

B. Rejection on the Merits

1. Rejections under 35 U.S.C. 103

Claims 13-37 are rejected under 35 U.S.C.103 (a) as being as unpatentable over *Vedula* et al. (US Patent No. 7,159,185 B1) in view of *Sindhu* et al. (US Patent No. 6,917,620).

Vedula teaches graphically creating a mapping between a source object and a target object. *Vedula* teaches:

> The source and target objects 26 and 30, respectively, may comprise XML or other schemas, spreadsheets, documents, databases, and/or other information or data sources. The user interface 20 may accordingly be adapted to display the objects 26 and 30 in a hierarchical tree structure to allow for ease of mapping creation. Once a mapping (e.g., mapping 44) has been created in the mapping screen region 42, a user may invoke a compiler (not shown) in the mapping tool, which generates compiled output code which may then be used in a runtime engine to translate source documents into target or destination documents. The output code will advantageously include the computer-executable instructions 6 from the script component 4, to

perform the function associated with the function object graphical component 8.

> Referring now to FIG. 3, an application of the invention is illustrated schematically, wherein a system 50 includes a source XML document 52, a target XML document 54, with an XSL engine 56 therebetween. The XSL engine 56 may comprise, for example, a computer system, which provides data transformations between the source XML document 52 and the target XML document 54 in accordance with an XSLT map 58 generated graphically in accordance with the invention. In this regard, the graphical user interface 20 of FIG. 2 may be used to create a mapping (not shown), which is then compiled into the computer executable instructions or codes, for example, XSLT code (e.g., map 58), and run by the engine 56 in performing the data transformation. Although the source and target documents 52 and 54 are illustrated as comprising XML information, the invention finds application with other forms of source and/or target information. Further in this regard, the output code may, but need not, be XSL or XSLT, whereby it will be appreciated that other output code languages fall within the scope of the invention.

Vedula, col. 9, ll. 22-56 (emphasis added). The disclosure of *Vedula* is focused on compiling mapping code to be used to translate source documents into target documents. The above disclosure from *Vedula* broadly references the source object and target object and the fact that the mapping can be used by a run-time engine. However, *Vedula* does not teach specifically how data of the source/target object is stored. The broad statement "The source and target objects 26 and 30, respectively, may comprise XML or other schemas, spreadsheets, documents, databases, and/or other information or data sources," *id.*, simply does not provide enough teachings for one of skill in the art to be able to implement the system of claim 13.

Claim 13 recites a specific manner for using a high-performance run-time engine to dynamically generate in-memory data components for storing actual data related to arbitrarily defined data structures. For example, Claim 13 recites:

> dynamically generate a first [or second] in-memory data component containing [or configured to store] actual data associated with the leaf data elements of the first [or second] data structure description, the high-performance run-time engine using a key-based look-up molding technique to generate a unique key to dynamically store and locate the actual data for each of the leaf data elements, the first [or second] in-memory data component comprising at least one lookup table configured to store each unique key for the first [or second] in-memory data component

Vedula simply does not teach or suggest dynamically generating in-memory data components in the specific manner recited in claim 13.

The Office Action indicated, and Applicant agrees, that *Vedula* does not disclose "traverse the one or more mapping descriptions and accessing the at least one lookup table of the first in-memory data component to get actual data stored in the first in-memory data component using a key-based look-up technique" as recited in claim 13. The Office Action asserted that this limitation is taught by *Sindhu*.[1]

Sindhu teaches a packet switching system, as follows:

> In operation, packets are received at a multi-function multiport 150, transferred to input switch 100 and stored temporarily in global data buffer 104. When the packet is received by switch 100, a key is read from the first data block in the packet and transferred to controller 106. The key contains destination information which is derived from the header field associated with the first block of data in a packet and other information (such as source ID, priority data and flow ID).

Sindhu, col. 8, ll. 30-37 (emphasis added).

Sindhu further teaches that when a packet arrives at a multi-function multiport, the packet is divided into fixed length cells, each cell having a cell header and cell data and stored in memory. The cell header includes a type field, stream field, packet header fields, and an independent read request in the form of a multi-function multiport identifier and address. *Id.* at col. 9, ll. 12-27.

When a cell is received at the input switch, a round robin data handler transfers the cell to be stored in a memory bank in global data buffer. *Id.* at col. 10, ll. 24-27. The round robin data handler also includes a key reading engine for determining the key information associated with a first cell in a packet and a linking engine for linking cells in the same packet. *Id.* at col. 11, ll. 11-14. After the key is determined for a given packet, it is stored temporarily in key buffer in input switch until the entire packet has been stored in global data buffer. *Id.* at col. 11, ll. 15-18. The key is also sent to the controller. *Id.* at col. 16, ll. 54-57. At the controller, *Sindhu* teaches:

> When the controller receives the key information, it must determine a key type. In one implementation, a plurality of key types are defined. The user may define up to 4 types of keys, each having variable length. The key type can be defined by a two bit field in the header. A look-up of the two bit field is used to determine an appropriate trie to search.
>
> Thereafter, an assigned route look-up engine 110 performs a trie based search for the best variable length match of the key, with each key type defining a particular

1 By arguing that the claims are distinguishable over *Sindhu*, Applicant does not concede that *Sindhu* is analogous art and reserves the right to dispute that *Sindhu* is analogous art.

> trie for searching. A trie is a data structure that is used to locate the best (longest) matching route for a given key. At the completion of the trie search, the route look-up engine returns a result which includes the output port associated with the destination. The result and other information (source ID, flow ID, packet length, quality of service and statistical information) for routing the packet through the router combine to form a notification. The notification is transferred from the controller 106 to the output switch 102. Upon receiving the notification, the output switch 102 initiates the transfer of the packet from memory 104 to the respective output port 150 associated with the result.

Id. at col. 17, ll. 6-26 (emphasis added).

The Office Action indicated that it would be obvious to modify *Vedula* with teachings from the claims of *Sindhu* relating to the controller key look-up engine and route memory. Specifically, the Office Action cited the following language:

> a controller coupled to the input switch, the controller including a key look-up engine and a route memory, the route memory storing a route table where the route table includes a trie, the key look-up engine traversing the trie to determine a best match to the key, and upon determining the best match for the key

Id. at claim 11. However, the key in *Sindhu* is used to identify a destination of a data packet. The key in *Sindhu* is stored in a routing table to determine a matching route for routing the packet. As discussed in the Telephone Interview, the manner and purpose of using the keys taught in *Sindhu* is very different than the use of keys in the present invention. Thus, *Sindhu* does not teach using keys to "dynamically store and locate the actual data for each of the leaf data elements" as recited in claim 13. Rather, *Sindhu* teaches using keys to locate a destination route.

Furthermore, the key in *Sindhu* is hard-coded into the data packet before arriving at the switch. *Sindhu* does not teach that the key is generated as recited in independent claim 13. One advantage of the present invention that it allows keys to be generated for arbitrarily defined data structures. See Specification, para. 33. Thus, even if *Vedula* were modified to incorporate the switching functionality taught by *Sindhu*, the combination still does not teach generating a unique key to dynamically store and locate the actual data for each of the leaf data elements as recited in independent claim 13. As such, Applicant respectfully requests that the obviousness rejection with respect to claim 13 be withdrawn.

For much the same reasons, the combination of *Vedula/Sindhu* does not teach or suggest the limitations of claim 31. That is, the combination of *Vedula/Sindhu* does not teach or suggest:

> set data values of any leaf data element of the first data structure description in a first in-memory data component, the first in-memory data component being dynamically generated using a key generated from a key-based look-up molding technique, wherein the key is stored in a lookup table associated with the first in-memory data component
>
> get the data values of any leaf data element of the first data structure from the first in-memory data component using the key generated from the key-based look-up molding technique

As such, Applicant respectfully requests that the obviousness rejection with respect to claim 31 be withdrawn.

Finally, the above arguments apply to claim 20. In addition, the Office Action asserted that *Vedula* teaches "a key is generated containing a concatenation of all names of the data containers in a hierarchical path to the leaf data element." Applicant respectfully disagrees. The Office Action cited *Vedula* as teaching "an exemplary function object 100 is illustrated, having a script component 104 with computer-executable instructions 106 for performing a string concatenation function," *Vedula*, col. 9, ll. 66-67, col. 10, ll. 1-2. As discussed in the Telephone Interview, this section of *Vedula* refers to various mapping functions object that can be displayed as a graphical component to perform mapping functions. Since *Sindhu* does not teach generating keys (rather, in *Sindhu*, the keys are already hardcoded), it would not be obvious to combine the concatenation function taught in *Vedula* with the keys taught in *Sindhu*. As such, Application respectfully requests that the obviousness rejection with respect to claim 20 be withdrawn.

Dependent claims 14-19, 21-30 and 32-37 depend from independent claims 13, 20 and/or 31 and thus incorporate the limitations thereof. As such, Applicant respectfully submits that claims 14-19, 21-30 and 32-37 are distinguishable over the prior art for at least the same reasons discussed above with respect to claim 13, 20 and/or 31 and request that the obviousness rejection with respect to these claims be withdrawn.

C. Conclusion

In view of the foregoing, applicant respectfully requests the Examiner's consideration and allowance of claims 13-37 as presented herein.

Applicant notes that this response does not discuss every reason why the presented claims are distinguished over the cited prior art. Most notably, applicant submits that many if not all of the dependent claims are independently distinguishable over the cited prior art. Applicant has merely

submitted those arguments which it considers sufficient to clearly distinguish the claims over the cited prior art.

In the event that the Examiner finds remaining impediment to a prompt allowance of this application that may be clarified through a telephone interview, the Examiner is requested to contact the undersigned attorney.

Dated this 4th day of January, 2008.

Respectfully submitted,

/sara d. jones/ Registration # 47,691

SARA D. JONES
Registration No. 47,691
Attorney for Applicant
Customer No. 022913

SDJ:vlr